AIND Term 1
Planning

By: Andika Tanuwijaya

Introduction

Planning is a problem to devise a plan of actions to achieve goals. The problem can be solved by using search-based or hybrid logical agent. In this project, we are trying to solve one example of the planning problem by using a representation for planning problems that scales up to problems that could not be handled by those two previous approaches. The problem is represented using a simple Planning Domain Definition Language that describes the problem definition: initial state, available actions, action result, and goal test. In this project, observation and comparison is performed on several uninformed search algorithms and several search algorithms with heuristics that are used to solve 3 variations of a planning problem.

Observation

In the observation performed, a time limit of 10 minutes is used to limit each algorithm execution, thus algorithms that exceed the time limit have no observation results. Following is the detail of problem variations and search algorithm used:

Problems:

- Problem A
  - Init(At(C1, SFO) ∧ At(C2, JFK) ∧ At(P1, SFO) ∧ At(P2, JFK) ∧ Cargo(C1) ∧ Cargo(C2) ∧ Plane(P1) ∧ Plane(P2)∧ Airport(JFK) ∧ Airport(SFO))
  - Goal(At(C1, JFK) ∧ At(C2, SFO))
  - Optimal solution example:
    - Load(C1, P1, SFO)
    - Load(C2, P2, JFK)
    - Fly(P1, SFO, JFK)
    - Fly(P2, JFK, SFO)
    - Unload(C1, P1, JFK)
    - Unload(C2, P2, SFO)
- Problem B
  - Init(At(C1, SFO) ∧ At(C2, JFK) ∧ At(C3, ATL) ∧ At(P1, SFO) ∧ At(P2, JFK) ∧ At(P3, ATL) ∧ Cargo(C1) ∧ Cargo(C2) ∧ Cargo(C3)∧ Plane(P1) ∧ Plane(P2) ∧ Plane(P3) ∧ Airport(JFK) ∧ Airport(SFO) ∧ Airport(ATL))
  - Goal(At(C1, JFK) ∧ At(C2, SFO) ∧ At(C3, SFO))
  - Optimal solution example:
    - Load(C1, P1, SFO)
    - Load(C2, P2, JFK)
    - Load(C3, P3, ATL)
    - Fly(P1, SFO, JFK)
    - Fly(P2, JFK, SFO)
    - Fly(P3, ATL, SFO)
    - Unload(C3, P3, SFO)
    - Unload(C1, P1, JFK)
    - Unload(C2, P2, SFO)
- Problem C

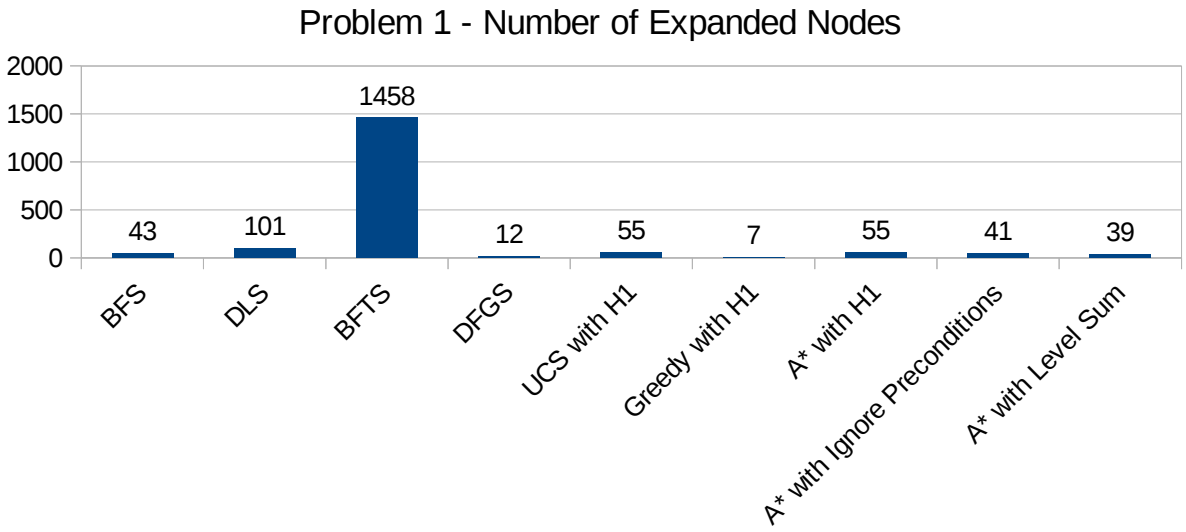
- Init(At(C1, SFO) ∧ At(C2, JFK) ∧ At(C3, ATL) ∧ At(C4, ORD) ∧ At(P1, SFO) ∧ At(P2, JFK) ∧ Cargo(C1) ∧ Cargo(C2) ∧ Cargo(C3) ∧ Cargo(C4) ∧ Plane(P1) ∧ Plane(P2) ∧ Airport(JFK) ∧ Airport(SFO) ∧ Airport(ATL) ∧ Airport(ORD))
- Goal(At(C1, JFK) ∧ At(C3, JFK) ∧ At(C2, SFO) ∧ At(C4, SFO))
- Optimal solution example:
  - Load(C1, P1, SFO)
  - Load(C2, P2, JFK)
  - Fly(P1, SFO, ATL)
  - Load(C3, P1, ATL)
  - Fly(P2, JFK, ORD)
  - Load(C4, P2, ORD)
  - Fly(P2, ORD, SFO)
  - Fly(P1, ATL, JFK)
  - Unload(C4, P2, SFO)
  - Unload(C3, P1, JFK)
  - Unload(C1, P1, JFK)
  - Unload(C2, P2, SFO)

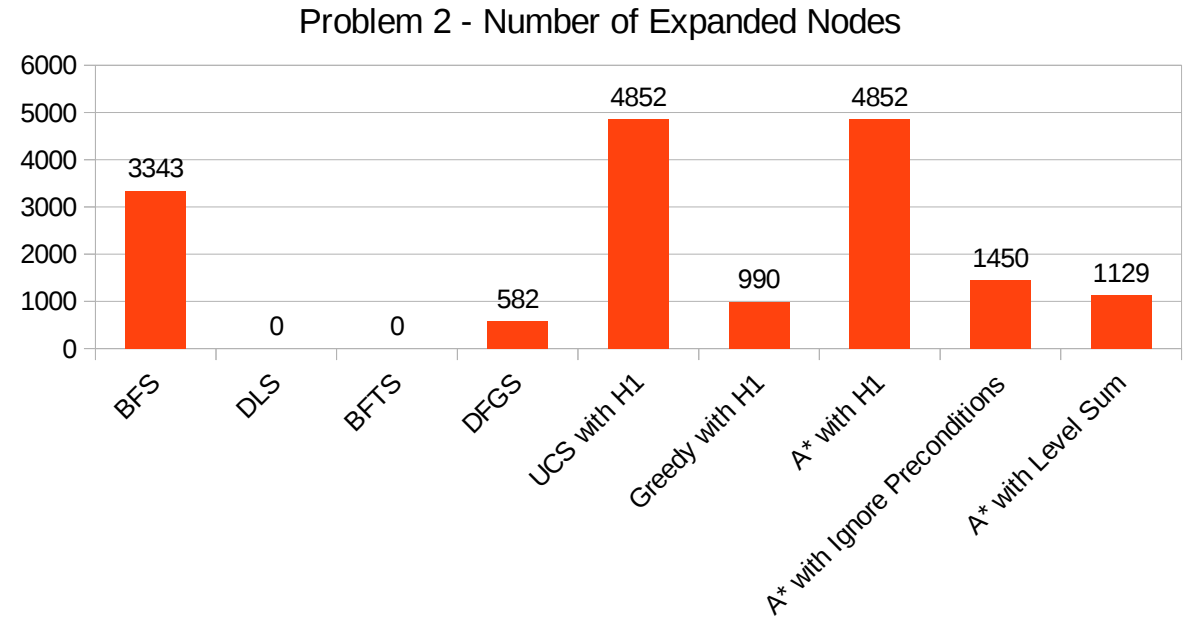
Algorithms:

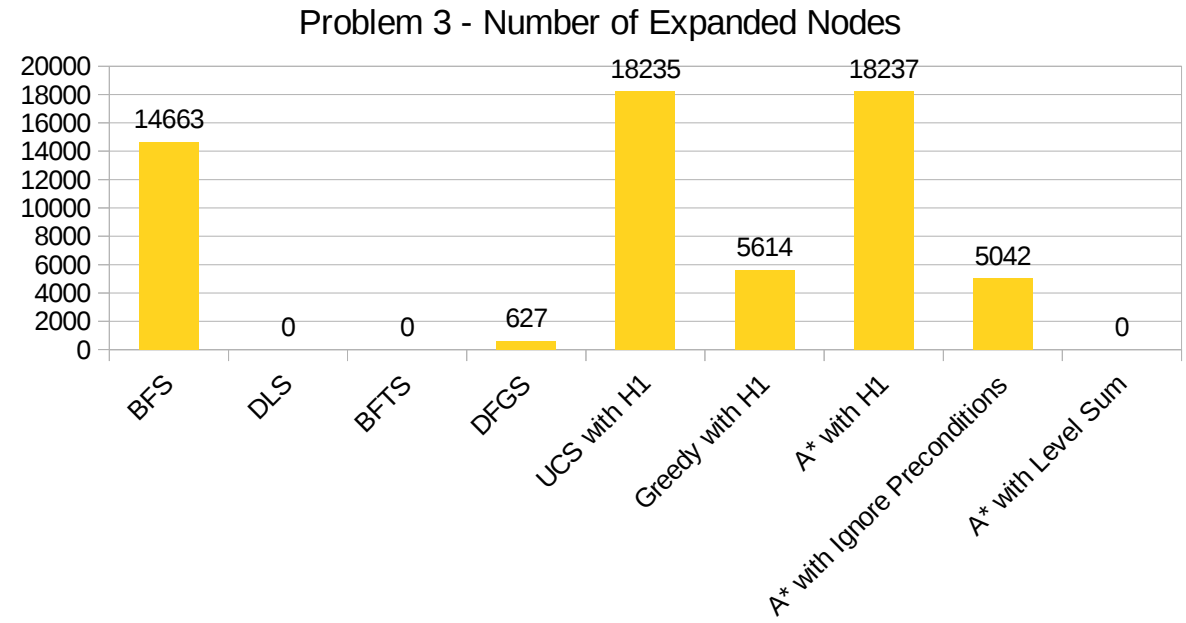
- Algorithm 1 – Breadth First Search
- Algorithm 2 – Depth Limited Search
- Algorithm 3 – Breadth First Tree Search
- Algorithm 4 – Depth First Graph Search
- Algorithm 5 – Uniform Cost Search with Constant Heuristics
- Algorithm 6 – Greedy Best First Graph Search with Constant Heuristics
- Algorithm 7 – A* Search with Constant Heuristics
- Algorithm 8 – A* Search with Ignore Preconditions Heuristics
- Algorithm 9 – A* Search with Level Sum Heuristics

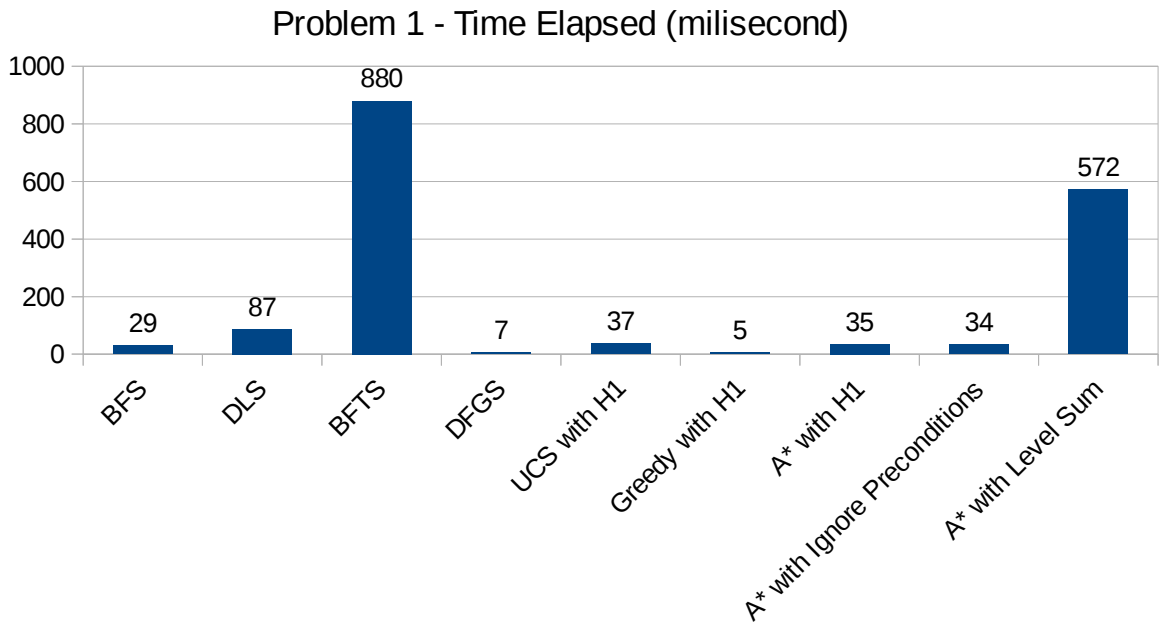
Observation Result

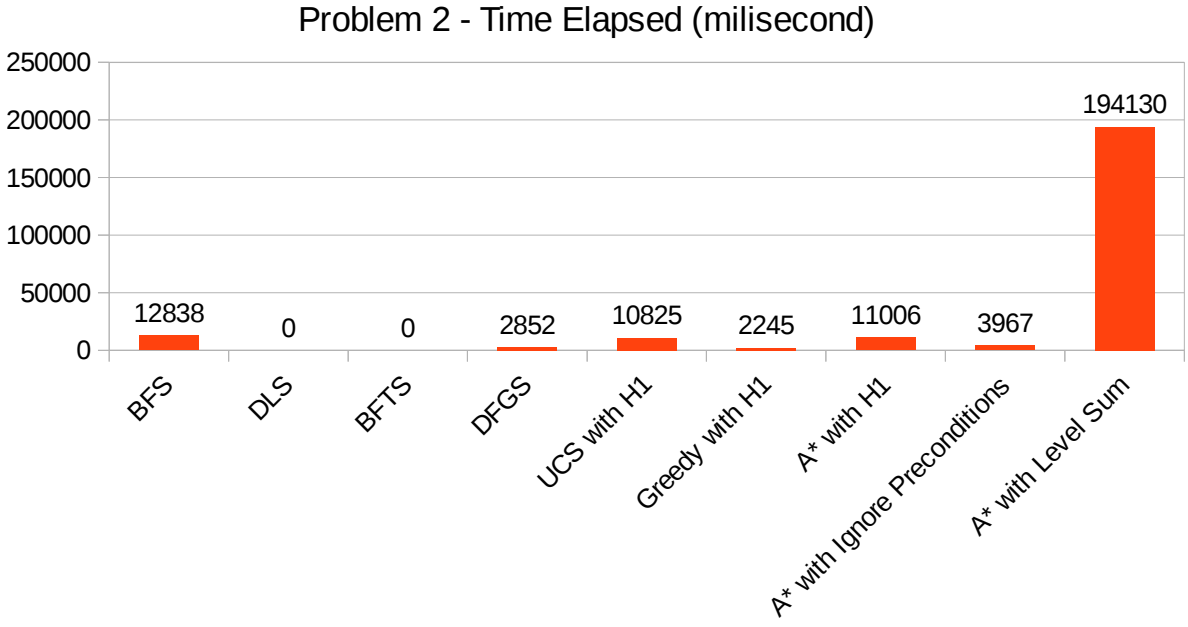
Problem	Algorithm	Number of Expanded Nodes	Number of Goal Tests	Number of Time Elapsed (second)	Optimality of solution
Problem 1	BFS	43	56	0.029	optimal
	DLS	101	271	0.087	less optimal
	BFTS	1458	1459	0.880	optimal
	DFGS	12	13	0.007	less optimal
	UCS with H1	55	57	0.037	optimal
	Greedy with H1	7	9	0.005	optimal
	A* with H1	55	57	0.035	optimal
	A* with Ignore Preconditions	41	43	0.034	optimal
	A* with Level Sum	39	41	0.572	optimal
Problem 2	BFS	3343	4609	12.838	optimal
	DLS	N/A	N/A	N/A	N/A
	BFTS	N/A	N/A	N/A	N/A
	DFGS	582	583	2.852	less optimal
	UCS with H1	4852	4854	10.825	optimal
	Greedy with H1	990	992	2.245	less optimal
	A* with H1	4852	4854	11.006	optimal
	A* with Ignore Preconditions	1450	1452	3.967	optimal
	A* with Level Sum	1129	1131	194.130	optimal
Problem 3	BFS	14663	18098	92.906	optimal
	DLS	N/A	N/A	N/A	N/A
	BFTS	N/A	N/A	N/A	N/A
	DFGS	627	628	2.968	less optimal
	UCS with H1	18235	18237	47.437	optimal
	Greedy with H1	5614	5616	14.735	less optimal
	A* with H1	18235	18237	47.673	optimal
	A* with Ignore Preconditions	5040	5042	15.330	optimal
	A* with Level Sum	N/A	N/A	N/A	N/A

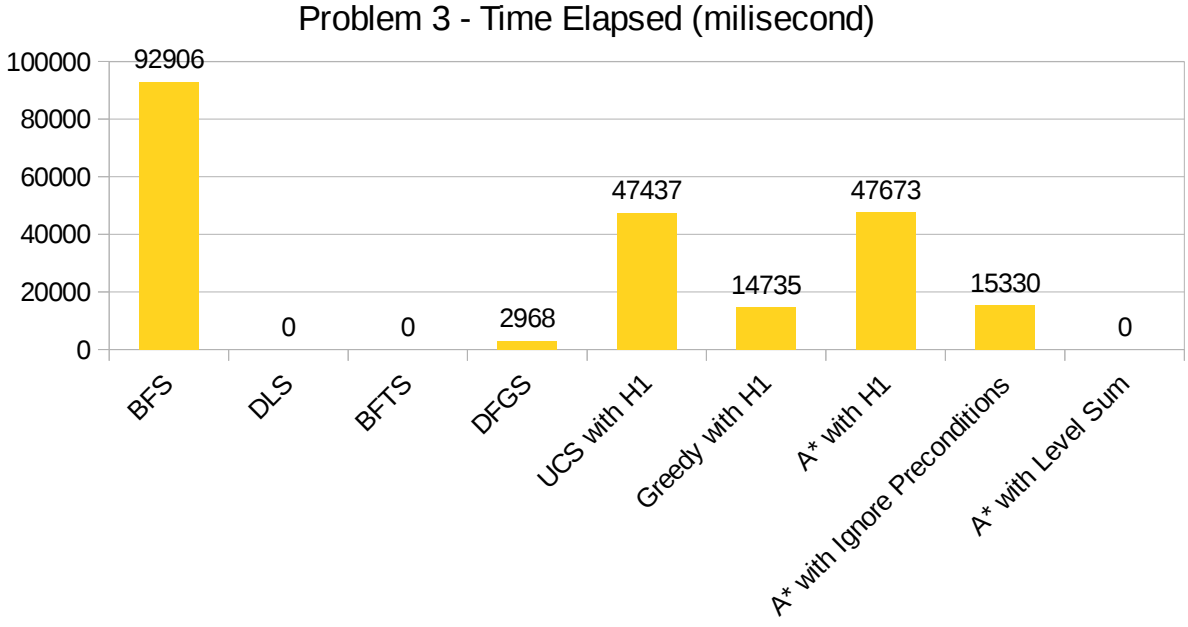

Problem 1 - Number of Expanded Nodes
2000
1500
1000
500
0
43
101
1458
12
55
7
55
41
39
BFS
DLS
BFTS
DFGS
UCS with H1
Greedy with H1
A* with H1
A* with Ignore Preconditions
A* with Level Sum


Problem 2 - Number of Expanded Nodes
6000
5000
4000
3000
2000
1000
0
3343
0
0
582
4852
990
4852
1450
1129
BFS
DLS
BFTS
DFGS
UCS with H1
Greedy with H1
A* with H1
A* with Ignore Preconditions
A* with Level Sum


Problem 3 - Number of Expanded Nodes
20000
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
14663
0
0
627
18235
5614
18237
5042
0
BFS
DLS
BFTS
DFGS
UCS with H1
Greedy with H1
A* with H1
A* with Ignore Preconditions
A* with Level Sum


Problem 1 - Time Elapsed (milisecond)
1000
800
600
400
200
0
29
87
880
7
37
5
35
34
572
BFS
DLS
BFTS
DFGS
UCS with H1
Greedy with H1
A* with H1
A* with Ignore Preconditions
A* with Level Sum


Problem 2 - Time Elapsed (milisecond)
250000
200000
150000
100000
50000
0
12838
0
0
2852
10825
2245
11006
3967
194130
BFS
DLS
BFTS
DFGS
UCS with H1
Greedy with H1
A* with H1
A* with Ignore Preconditions
A* with Level Sum


Problem 3 - Time Elapsed (milisecond)
100000
80000
60000
40000
20000
0
92906
0
0
2968
47437
14735
47673
15330
0
BFS
DLS
BFTS
DFGS
UCS with H1
Greedy with H1
A* with H1
A* with Ignore Preconditions
A* with Level Sum

By observing the result of problem 1, most algorithm perform similarly with exception of DLS and DFGS which produce less optimal solution for the given problem and BFTS which expands too much nodes with significant running time. Breadth first search is a recommended uninformed search algorithm for problem 1 with DFGS as runner-up if optimality is not a concern. Greedy with H1 and A* with ignore precondition are recommended as heuristics-based search algorithm.

In the result of problem 2, we can see that some of the working algorithms in problem 1 have poor performance in the larger problem. In problem 2, DLS and BFTS run more than the specified timeout, DFGS and Greedy search expand a minimal number of nodes but produce less optimal solutions, while A* with level sum heuristics performs in hefty running time. BFS is still recommended for problem 1 as uninformed search and A* with ignore preconditions for the informed search counterpart.

In the larger problem 3, A* with level sum follows the previous 2 algorithms in exceeding the specified timeout while DFGS and Greedy also produce less optimal solutions. BFS is recommended uninformed search in this problem if optimality is a concern, otherwise DFGS is a fast algorithm but produces less optimal solution. As for informed search algorithms, A* with ignore preconditions still dominates the competition.

Overall, all uninformed search algorithms have rather similar performance when the problem they are applied to is small, but when the problem is large BFS is a solid solution and DFGS as secondary option if optimality is not a concern. While A* algorithm is able to outperform other informed search algorithms depending on the heuristics used. This complies with the explanation by Peter Norvig in his book AIMA page 108-109 (3rd Ed).

Uninformed search methods have access only to the problem definition. The basic algorithms are as follows:

- **Breadth-first search** expands the shallowest nodes first; it is complete, optimal for unit step costs, but has exponential space complexity.
- **Uniform-cost search** expands the node with lowest path cost, $g(n)$, and is optimal for general step costs.
- **Depth-first search** expands the deepest unexpanded node first. It is neither complete nor optimal, but has linear space complexity. **Depth-limited search** adds a depth bound.
- **Iterative deepening search** calls depth-first search with increasing depth limits until a goal is found. It is complete, optimal for unit step costs, has time complexity comparable to breadth-first search, and has linear space complexity.
- **Bidirectional search** can enormously reduce time complexity, but it is not always applicable and may require too much space.

Informed search methods may have access to a **heuristic** function $h(n)$ that estimates the cost of a solution from n.

- The generic **best-first search** algorithm selects a node for expansion according to an **evaluation function**.
- **Greedy best-first search** expands nodes with minimal $h(n)$. It is not optimal but is often efficient.
- **A* search** expands nodes with minimal $f(n) = g(n) + h(n)$. A* is complete and optimal, provided that $h(n)$ is admissible (for TREE-SEARCH) or consistent (for GRAPH-SEARCH). The space complexity of A* is still prohibitive.
- **RBFS** (recursive best-first search) and **SMA*** (simplified memory-bounded A*) are robust, optimal search algorithms that use limited amounts of memory; given enough time, they can solve problems that A* cannot solve because it runs out of memory.